(2)

AL-TP-1991-0020

AD-A236 211

ARMSTRONG LABORATORY

HIERARCHICAL MODELING AND PROCESS AGGREGATION IN OBJECT-ORIENTED SIMULATION

Douglas A. Popken, Capt, USAF

HUMAN RESOURCES DIRECTORATE
LOGISTICS RESEARCH DIVISION
Wright-Patterson Air Force Base, OH 45433-6503

May 1991

Interim Technical Paper for Period January 1990 - March 1991

AIR FORCE SYSTEMS COMMAND
BROOKS AIR FORCE BASE, TEXAS 78235-5000

[illegible]

DOUGLAS A. POPKEN, Capt, USAF
Project Scientist

[illegible]

BERTRAM W. CREAM, Technical Director
Logisitics Research Division

[illegible]

JAMES C. CLARK, Colonel, USAF
Chief, Logisitics Research Division

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** May 1991	**3. REPORT TYPE AND DATES COVERED** Interim Paper – January 1990 – March 1991

4. TITLE AND SUBTITLE Hierarchical Modeling and Process Aggregation in Object-Oriented Simulation	**5. FUNDING NUMBERS** PE – 62205F PR – 1710 TA – 00 WU – 18
6. AUTHOR(S) Douglas A. Popken	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Armstrong Laboratory, Human Resources Directorate, Logistics Research Division, Wright-Patterson Air Force Base, OH 45433-6503	**8. PERFORMING ORGANIZATION REPORT NUMBER** AL-TP-1991-0020
9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT (Maximum 200 words)

The objective of this research was to develop techniques for more easily building simulation models with resolution levels appropriate for their purpose. The basic premise was that a modular, hierarchical modeling approach would be best suited to accomplish this and that object-oriented programming techniques would be a primary enabling technology. Within the proposed hierarchical modeling framework, submodels may be represented in either their original form or in an aggregated form, the choice being governed by the goals and constraints of a particular simulation study. In this paper a prototype hierarchical modeling system supporting aggregation is presented. The prototype was coded in the SMALLTALK-80 object-oriented programming environment, and is tailored to an airbase logistics application. The paper also discusses several possible approaches to aggregation, concluding that the "simulative approach" is most appropriate for this application.

14. SUBJECT TERMS aircraft maintenance, artificial intelligence, high-level languages, logistics planning, simulation	**15. NUMBER OF PAGES** 32
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

PREFACE

The research described in this paper was performed in-house at the Logistics and Human Factors Division of the Air Force Human Resources Laboratory. It was designed to support the Productivity Improvements in Simulation Modeling (PRISM) project, a long-term effort to enhance the Air Force's ability to evaluate, through simulation, how differing logistical support scenarios may constrain sortie generation capability. Currently, the primary research being done under PRISM is the Integrated Model Development Environment contract effort.

Programming assistance for the SMALLTALK-80™ based prototype described in this paper was provided by Michelle Bagley of Systems Research Laboratories. Ms. Bagley was responsible for coding the "Network" objects and methods as well as the set of objects and methods needed to integrate the data from the simulative pre-analysis with the prototype simulation environment.

SUMMARY

The paper begins by introducing a modular, hierarchical approach for object-oriented simulation of complex problem domains. The advantages of this approach include enhanced modeling consistency across different levels of resolution and increased modeling productivity. A major part of the research was to allow submodels within the hierarchy, typically represented by networks, to be modeled in either their original form or in an aggregated form, the choice being determined by the goals and constraints of a particular simulation study. This capability would enhance the reuseability of models, foster better understanding of the processes being modeled, increase computational efficiency, and enable more selective recording of model outputs.

The paper then discusses several possible techniques for aggregating networks, including analytical techniques for product-form networks, approximate techniques, numerical techniques, and simulative analysis. The simulative aggregation technique was chosen for the prototype simulation environment for two main reasons: the need to extract complex data from the network being aggregated, and the desire to tightly couple the network aggregation/analysis with the simulation environment.

Lastly, the paper outlines the design of a prototype hierarchical modeling system. The prototype was coded in the SMALLTALK-80™ object-oriented programming environment and is tailored to an airbase logistics application. The application focuses on base-level maintenance of aircraft and their constituent parts. The sequences of maintenance activities (task networks) for either individual parts or the aircraft may be aggregated.

Hierarchical Modeling and Process Aggregation in Object-Oriented Simulation

I. INTRODUCTION

Discrete-event simulations typically model a problem domain in terms of "active" simulation entities whose behavior may be described in terms of various activities in time sequence. In this type of simulation, the possible sequences of activities for an active entity, its "flow of events," can be represented as an activity network. Each node in such a network represents an activity, while the arcs represent the possible transitions to succeeding activities (Figure 1). The actual transition made during the simulation may be dependent on any combination of logical or probabilistic preconditions. The activity has duration, possibly probabilistic, during which "passive" resources may be occupied, expended, or created, or during which interaction with other simulation entities may occur.

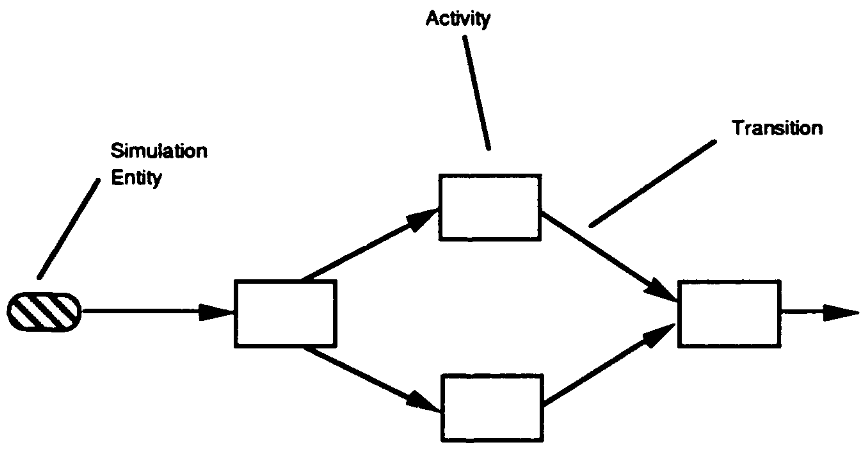


Figure 1. **Activity Network**

This network based "world view" has been implemented in simulation systems such as SIMNET II (Taha, Taylor, & Younis; 1990), Network II.5 (Garrison, 1990), and SLAM II (O'Reilly & Whitford, 1990). It is also consistent with the "process-oriented" approach to discrete-event simulation described in Franta (1977). In the process-oriented approach, the entity and its associated activity network are considered a "process" which competes with other processes for simulation resources.

Activity networks can be organized into hierarchies to simplify modeling of complex problem domains. The levels of the hierarchy would then represent different levels of problem resolution, with the topmost level representing the least detail, and the bottom level representing the greatest detail. As shown in Figure 2, what is a network at one level, is represented as a single node at the next higher level.

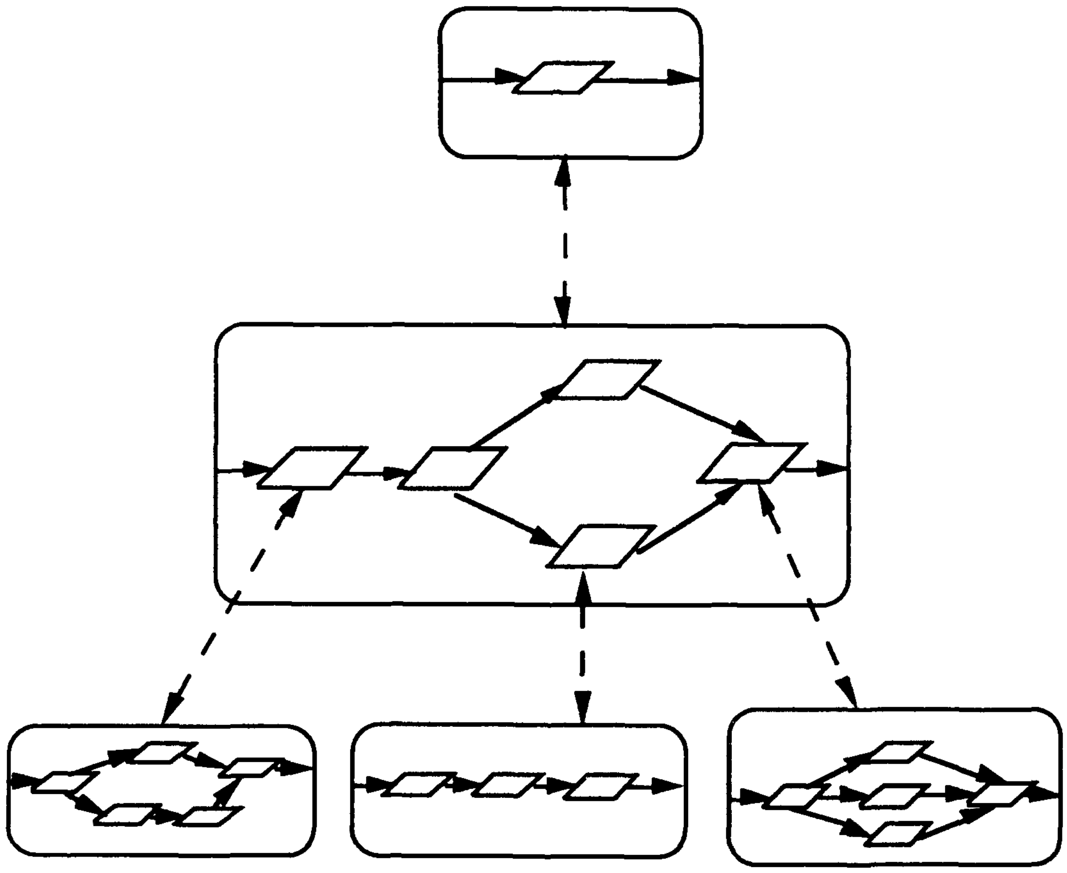


Figure 2. A Hierarchy of Activity Networks

The advantages of a hierarchical approach are enhanced modeling consistency across different levels of resolution and increased modeling productivity. Productivity increases arise from the ability to use modular, pretested components (networks) in model development. Simulations may then be rapidly reconfigured for use as *prescriptive* models of design alternatives or *descriptive* models of differing aspects of a single given problem domain. The theoretical foundations for the modular, hierarchical simulation approach may be found in Ziegler (1984).

This paper describes an object-oriented implementation of a modular hierarchical modeling framework. Object-oriented software provides support for hierarchical structuring of model components through both class-subclass and part-whole relationships (Coad & Yourdon, 1990). In the approach to be described, the networks and their constituent nodes are represented as "objects." The *encapsulation* property of object-oriented programming provides a way to achieve the desired modularity. The object-oriented property of *polymorphism* reinforces this notion by allowing active simulation entities to interact with nodes (aggregated networks) or networks identically, that is, without knowledge of their actual internal structure.

The paper also describes the relationships between a process (network) and its corresponding higher level representation (node). As in Fishwick (1986), we distinguish between two types of relationships: *aggregation* and *abstraction*. In aggregation, the assumption is made that detailed knowledge about a process (i.e., its lowest level representation in the hierarchy) is already known. Mathematically based techniques are then used to obtain an aggregated version of the process which is less detailed than but mathematically consistent with--at least in an approximate sense--the original version of the process. The features of a network that must be aggregated will generally depend on the problem domain. However, we assume here that resource interactions strictly within the network are not retained; otherwise we might just as well use the detailed network model. In general, aggregation will be feasible as long as meaningful consistency criteria are maintained between levels of aggregation (Rothenberg, Narain, Steeb, Hefley, & Shapiro; 1989). For many problems, it suffices to obtain an aggregate measure of the traversal time of a network by a simulation entity. Section II will describe a general approximate technique for temporal aggregation of an activity network via preprocessing activities; that is, there is no "automatic" aggregation at run time.

In abstraction, on the other hand, we derive the higher level representation from observed or known high-level system characteristics rather than from mathematical transformations upon the detailed simulation. This approach allows the layers to differ both conceptually, and in their representation (Fishwick, 1986). While the software implementation of this approach is fully supported by the hierarchical framework described here, specific abstraction techniques tend to be application-specific and are beyond the scope of this paper.

The purpose of having the capability to abstract or aggregate is to provide simulation models whose resolution is appropriate for the problem being studied. This would enhance the reusability of models, foster better understanding of the processes being modeled, increase computational efficiency, and enable more selective recording of simulation outputs.

As an illustration, consider a modeler who is only concerned with some subset of a simulation entity's possible activities within a given problem domain. If we think of the hierarchical framework as a union of all relevant ways to model the sequence of activities a simulation entity may traverse, then selecting the design of a particular simulation model implementation is analogous to choosing a particular path through the hierarchy. Path selection is defined here as determining for each node whether a simulation entity will traverse the node or its more detailed (lower level) representation. The design (path) is chosen by the designer according to the type of information the simulation is to present. While complex simulations may have some processes modeled at a detailed level with others modeled at a higher level of abstraction, the level(s) of detail is(are) typically decided early in the design phase and can only be varied later with a great deal of reprogramming effort. Obviously, the mixture of modeling detail levels appropriate for one study may be totally inappropriate for another. In such cases it would be desirable to have the ability to select a design (path) from an existing framework, rather than designing an entirely new simulation model. The path would be selected according to the circumstances (criticality, time constraints, available computing resources, etc.) of the particular study.

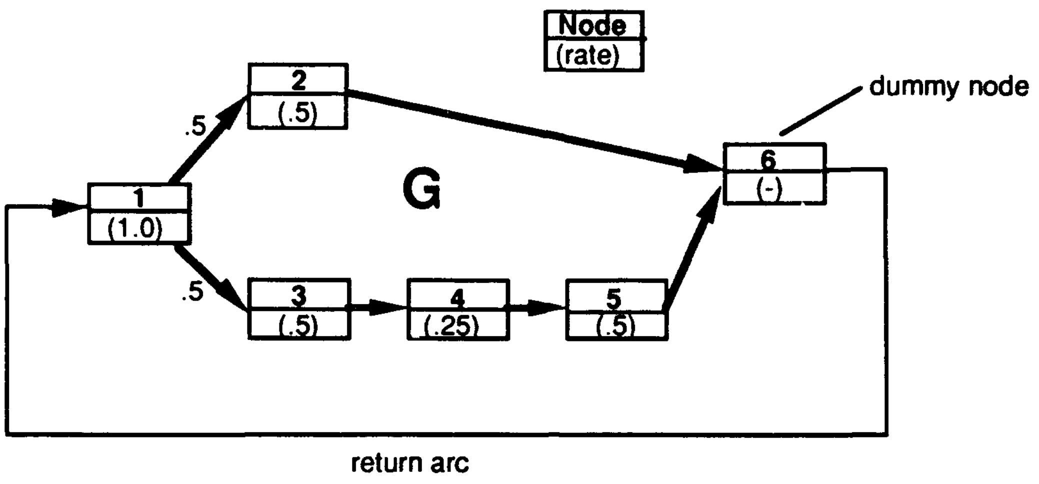
II. AGGREGATION TECHNIQUE

A key difficulty in network aggregation is ensuring the time required for an entity to traverse the aggregated network is approximately the same as it would be in the original network. This is especially true if we seek to retain in the aggregate an entire network traversal time distribution and not just its mean. Several factors make this difficult. First, we must account for the affects of congestion and the resulting waiting times on network traversal times. In the absence of congestion, the network traversal time distribution could otherwise be computed as a mathematical convolution of the service time distributions at each node. Second, we make no prior simplifying assumptions about the form of the probability distributions at the nodes; this precludes the use of powerful analytical techniques applicable only to special classes of networks and distributions. Before addressing these difficulties, we will discuss the possible approaches to aggregation and show how they relate to the approach of this paper.

There are four basic approaches to network aggregation (Buchholz, 1989): 1) analytical techniques for product form queuing networks, 2) approximate techniques, 3) numerical techniques, and 4) simulative analyses.

Product form queuing networks (i.e., networks in which the state space satisfies local balance) can be aggregated into single composite queues. The theoretica'. basis for this aggregation, Norton's Theorem, was first extended to include queuing networks by Chandy, Herzog, & Woo (1975). The extended version of the theorem states that it is possible to aggregate a queueing network into a single composite queue with appropriate service rates such that the original network and the composite queue have identical affects on their environment. However, while in our application we can not assume product-form queuing networks, the technique for aggregating product-form queuing networks provides some insight into possible techniques for aggregating more general networks.

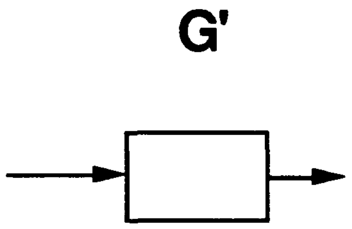
As an illustration of the aggregation of product-form networks, consider the closed network of queues, G (an open network can be converted into an equivalent closed network by adding "dummy nodes"--nodes having a service time of 0--and by adding a return arc from last node to first node), shown in Figure 3.



n =	1	2	3
Throughput rate =	.167	.273	.343

Figure 3. **Closed Queuing Network**[1]

Assume local balance is satisfied and service rates at each node are as given. Norton's theorem allows us to construct an equivalent composite queue, G' (Figure 4), with service rate T(n), where n is the number of customers in the composite queue (n = 0,1,2,...N). T(n) is set to the steady state service rate of network G when n customers are present.



Queue size =	1	2	3
Service rate =	.167	.273	.343

Figure 4. **Composite Queue**

[1] The throughput rates result from a state-space approach, where the state is defined as the number of entities at each node of the network. The marginal probability of each state was calculated using a Markov process simulator. The network throughput rate is found by calculating the expected transition rate on the return arc.

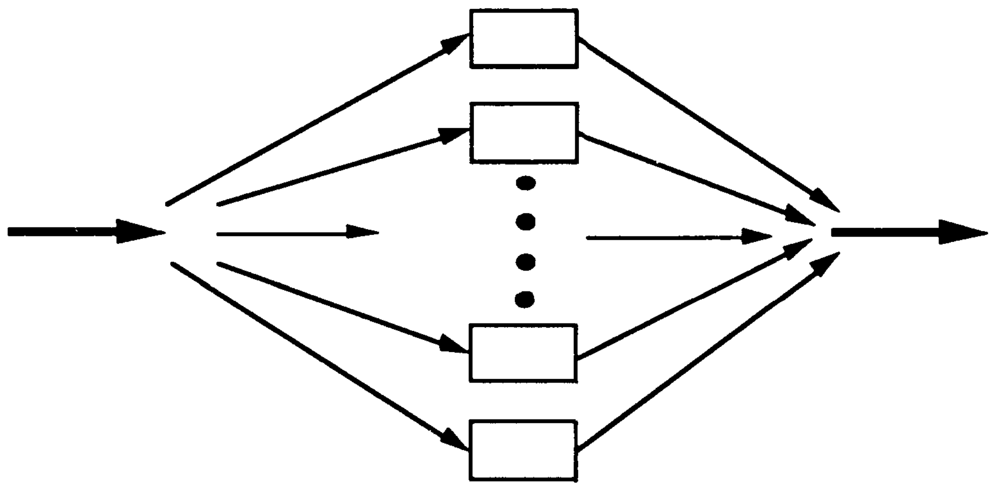
Approximate analytical techniques, including diffusion and iterative techniques, have been devised to analyze more general networks that do not satisfy local balance (Marie, 1979). For some classes of non-product form networks, approximate techniques have been shown to be relatively accurate and efficient. However, from a practical standpoint, the major drawback to these approaches is that they would require a large development effort to program sophisticated and complex algorithms that must also interface with the simulation environment. This would be complicated by the fact that computer languages are rarely suitable for both object-oriented simulation and complex mathematical analyses. However, it would be desireable to have any preprocessing analysis/aggregation tightly coupled with the simulation environment.

Numerical techniques are those that perform a numerical evaluation of the global balance equations (the input-output equations in the state space). Solving these equations produces the state probabilities. Numerical techniques require more computational effort than either product form or approximate techniques because they require solving a set of balance equations for each possible "population vector"--the vector describing the number of members in each possible class of active entities. In addition to the computational effort numerical techniques would also require a large development effort to integrate the preprocessing analysis with the simulation environment.

The final approach is simulative analysis of the network. This approach has several major drawbacks. Like the numerical techniques, simulative analysis requires finding a solution (running a simulation) for each possible population vector. In addition, simulative analysis, essentially a statistical technique, requires additional effort to interpret the statistical significance of results. On the other hand, simulative analysis has numerous advantages. A simulation can provide much more information than just the mean network traversal time. As will be described later in this paper, the entire probability distibution of the network traversal time may be extracted without additional computational effort. Also, there are no theoretical restrictions on the characteristics of the network that can be analyzed. Simulative analysis is the easiest to integrate with a simulation modeling environment.

We have chosen the simulative approach to aggregation. Under this approach, a simulation run consists of a closed network with a given population vector. A separate run is made for each possible population vector. If there is only one class of active entities traversing the network, the population vector is merely a single index (e.g., i) indicating the number of entities. Because the network is closed (Figure 3), each customer immediately returns to the beginning of the network after completing a traversal of the network. In this way there are always i customers traversing the network in a given simulation run. The time is recorded for each customer's traversal. Given sufficient simulation time, a discretized distribution of traversal times is built up for a given congestion level i. Note that the total traversal time is a sum of both service and waiting times.

After preprocessing, the aggregate network can be modeled in the simulation environment as an infinite server queue with service time distributions indexed by the number of serverן occupied (Figure 5). The distribution used when i servers are occupied is the discrete distribution resulting from the network simulation with congestion level i.



# servers occupied =	1	2	3
Traversal time dist. =	F_1	F_2	F_3

Figure 5. Infinite Server Queue

The advantage of this simulative approach is that at least an approximate version of the entire network traversal time distribution is preserved during the aggregation, even when the network does not satisfy local balance. Further, the network traversal time distribution is indexed by the state of the network. This state dependence is important, as congestion effects may radically alter the distribution of service times. As an example, Figure 6 illustrates the traversal time distribution for a version of the Figure 3 network (with general service time distributions) as the state variable (the number of customers present) is increased. In the case of one customer, the shape of the distribution is relatively well behaved. As congestion increases, variability and the bimodal shape of the distributions also increase. Bimodality is a consequence of having two possible paths through the network, each with its own characteristic affects on the traversing entity. As congestion increases, the difference in traversal times between the two paths grows steadily larger.

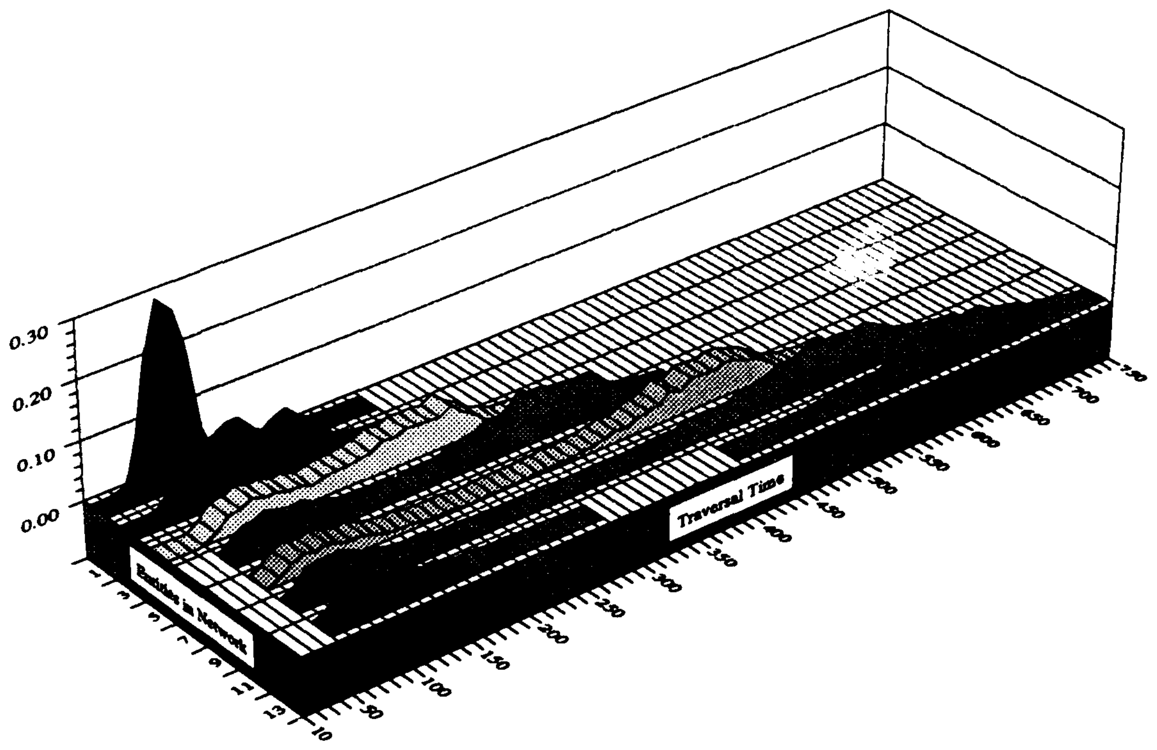


Figure 6. **Sample Traversal Time Distributions**

Figure 6 dramatically illustrates how, in some applications, one would be ill-advised to use an "expected value" approach to aggregation, where traversal times are represented by only the mean value. As congestion increases, the expected value becomes a less meaningful descriptor of throughput times. For example, in the case of $i = 13$, the sample mean is approximately 452, a value with little or no probability mass in its vicinity.

III. SOFTWARE IMPLEMENTATION OF THE HIERARCHICAL FRAMEWORK

A prototype of the modular hierarchical simulation framework described in this paper was developed using the SMALLTALK-80 programming environment (Goldberg and Robson, 1989). SMALLTALK provides a fully accessible library of object classes; in fact, the entire SMALLTALK environment is built on these classes. SMALLTALK also provides the classes necessary to implement the functionality of basic simulations (e.g., event timing and synchronization, queuing, and resource utilization). Developing the prototype framework required creating the necessary extensions to the basic class structure. These extensions included application-specific classes for an airbase logistics problem domain. Figures 7a through 7c illustrate an overview of the new simulation classes and how they relate to the basic structure. However, the figures only show those SMALLTALK classes that are part of the direct lineage of the new classes.

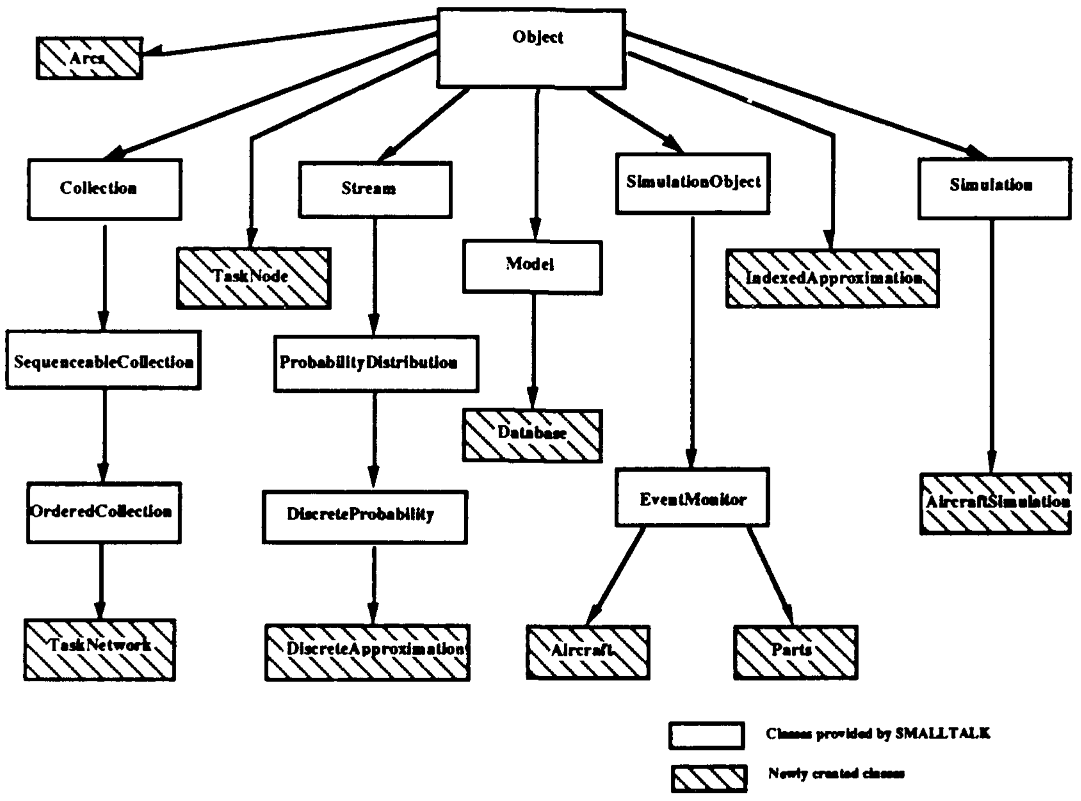


Figure 7a. Class Hierarchy of the Prototype Simulation System

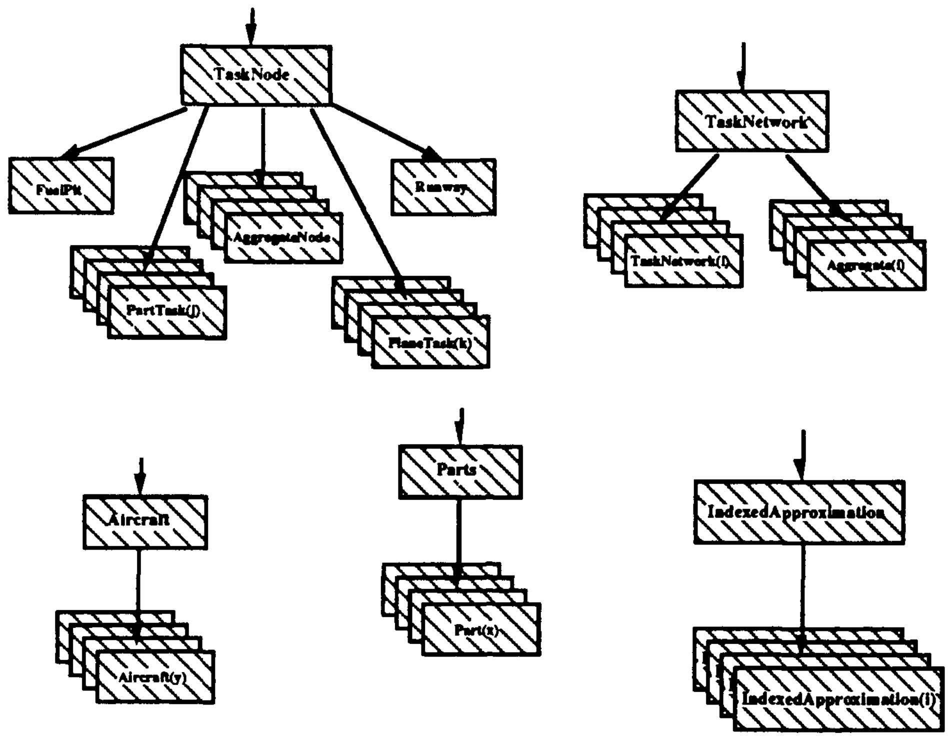


Figure 7b. Class Hierarchy of the Prototype Simulation System (Cont)

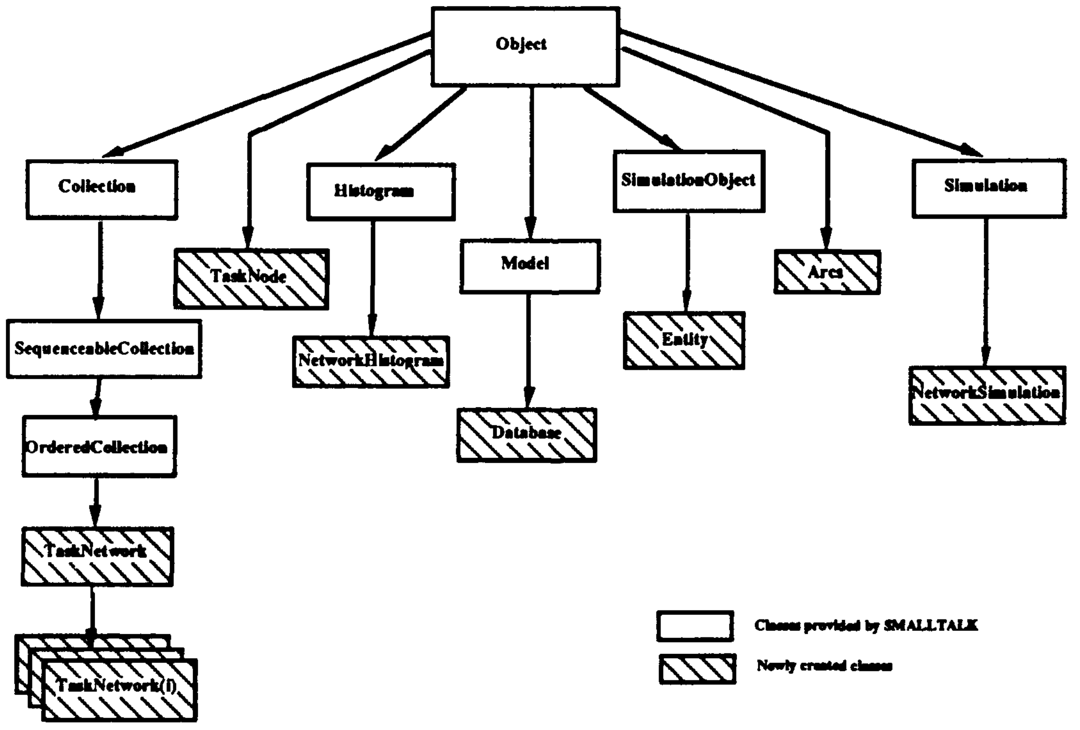


Figure 7c. Class Hierarchy of the Network Simulator

The active simulation entities are the subclasses of SimulationObject, generally speaking, Parts and Aircraft. These entities must cycle through sequences of activity networks representing various operational and maintenance related activities. The logic of the airbase logistics simulation is illustrated in Figure 8.

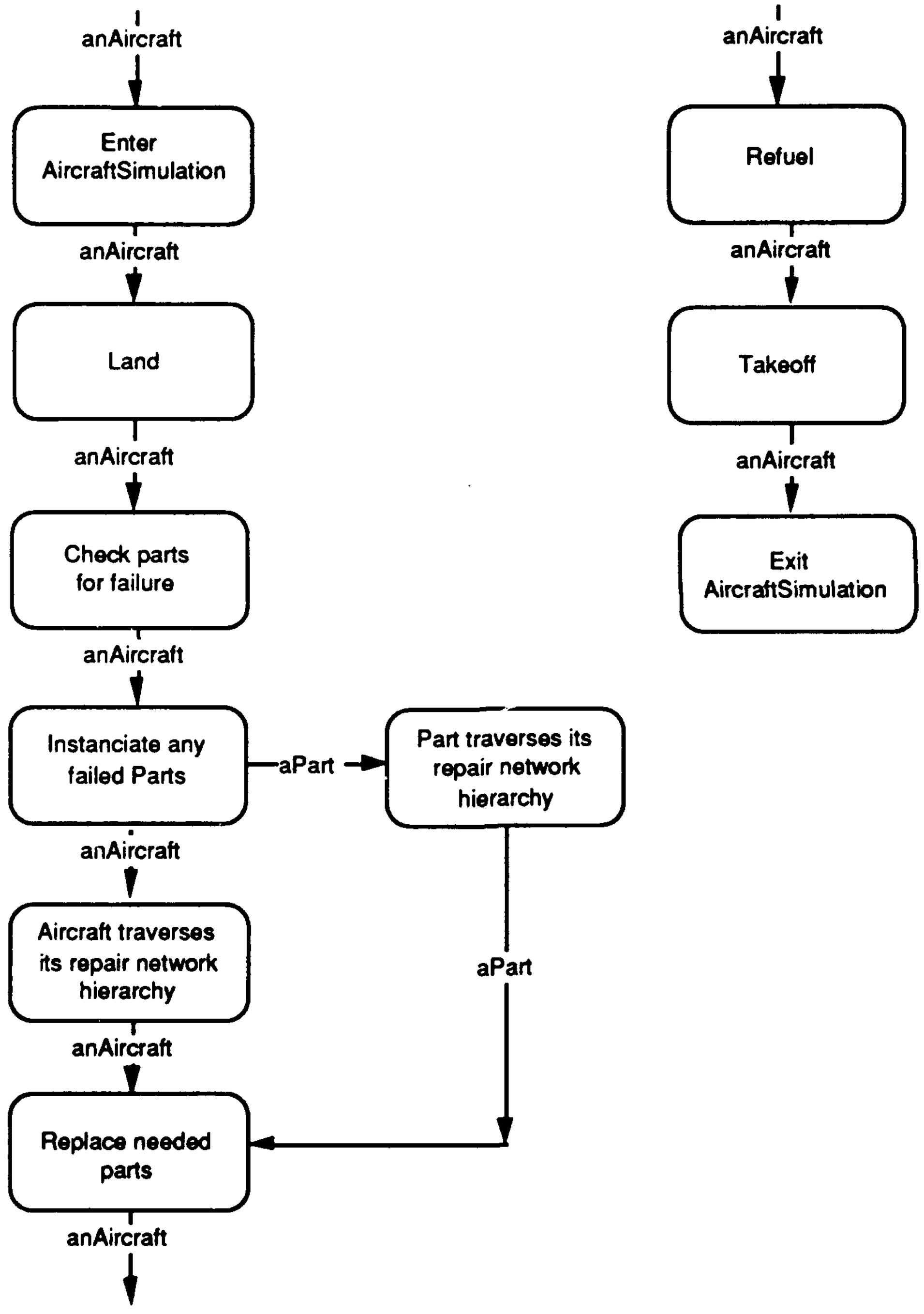


Figure 8. Logic of Airbase Logistics Simulation

The classes that provide the structure of the hierarchical framework are TaskNetwork, Arcs, and TaskNode. Instances of TaskNode are the focal points for simulated activities. As currently implemented, each node specifies the resources to be used, the length of time involved, and the possible transitions to succeeding activities. The nodes could also be used to specify logical conditions on resource usage or transition choices. Instances of Arcs provide the links to succeeding activities and the associated transition probabilities.

Each instance of TaskNetwork refers to a structured collection of instances of TaskNode and Arcs. In practice, this is achieved with a pointer to the "root node" of the network. The main purpose of TaskNetwork is to implement behaviors for network building, traversal, and information collecting.

Additional implementation details, including object attributes and key methods, may be found in Appendices A and B.

Network Preprocessing Simulation

Analyzing aggregate network behavior requires collecting data on the simulated network. The classes needed are the network related classes, specialized versions of existing basic SMALLTALK simulation classes, and a new persistent data management class, Database. In the prototype system, only network traversal time data is collected. The previous section of this paper pointed out the difficulty of describing these times in terms of ordinary probability distributions. To capture the form and shape of the distribution, an approximate approach is used in which throughput data is collected in histogram form. Data collection is implemented in a specialized Histogram class called NetworkHistogram. The Database class provides data persistence and manages the histogram data associated with each likely level of network congestion. The data for a particular congestion level is updated each time the Simulation subclass, NetworkSimulation, is run. The persistent data is then accessible by other simulations through the class IndexedApproximation, described in the next subsection. NetworkSimulation initializes the desired network object instances and manages the simulation process during which instances of the SimulationObject subclass, Entity, traverse the network. Typically, the simulation will be run long enough to collect data on several thousand traversals.

Hierarchical Simulation

To achieve the desired modularity, TaskNetwork has been provided with the same methods and variables as TaskNode. (If SMALLTALK supported multiple inheritance, the same effect could have been achieved by inheriting those features directly from TaskNode.) Thus, an instance of SimulationObject need not distinguish between node and network. Of course, the procedures activated upon receipt of a generic message will be quite different. In particular, the message *duration: aSimulationObject*, when received by an instance of TaskNode, causes the node to return either a fixed number or a random deviate generated by the probability distribution corresponding to the node's activity. However, when the message is received by an instance of TaskNetwork, a complex simulation process is invoked. The object specified by aSimulationObject will now traverse the receiving task network, returning to the original upper level only after completing the lower level. The approach places no limits on the levels of nesting of activity networks. An example of a hierarchy from the airbase logistics problem domain is shown in Figure 9.

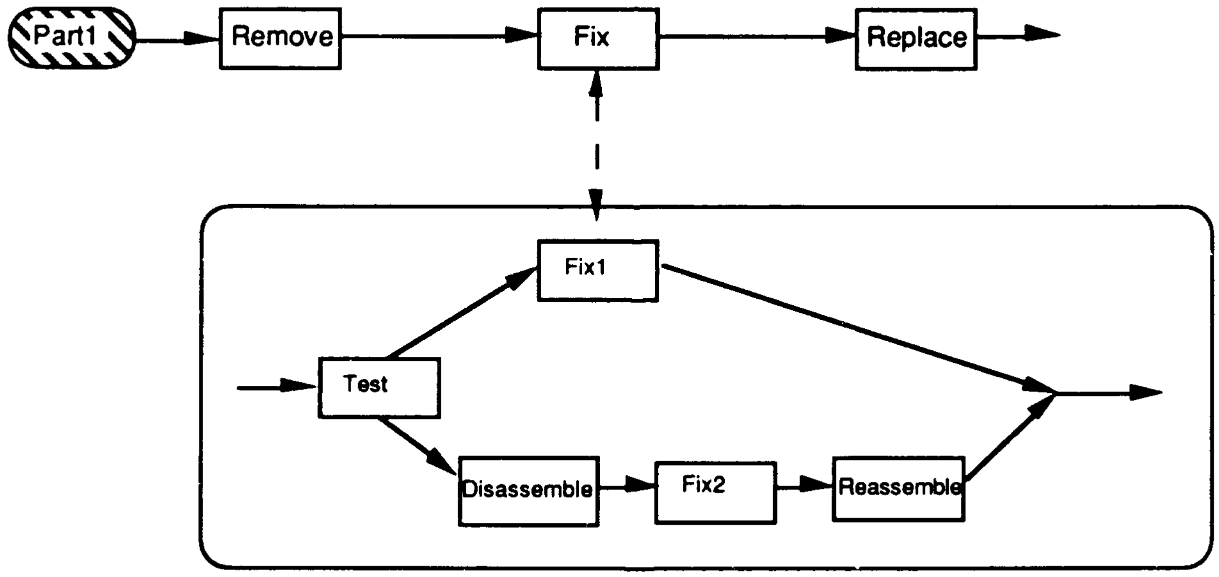


Figure 9. Hierarchy Example

Aggregated Networks

This case is the mirror image of the above: rather than having a network appear to the simulation as a node, the node will appear as a network. The nodes used are members of the Aggregate class, a subclass of TaskNode. Instances of Aggregate will have a special type of "dummy" resource and "duration" specified by an instance of the class, IndexedApproximation.

The dummy resource differs from ordinary Resource objects (provided by SMALLTALK) in that any number of simulation objects may be using them simultaneously. Thus they operate similarly to an infinite server queue. This provides a way to attach a simulation object to a "network" for the appropriate duration without precluding other simulation objects from doing so. This is appropriate in that the network itself is not a limited resource, but merely a conceptual aggregation of the actual resources.

Externally, instances of IndexedApproximation operate in the same manner as a probability distribution, that is, sending an instance the message *next* causes it to return a random deviate representing the aggregate node's duration (network traversal time). To account for network congestion, on receipt of *next*, it will query the dummy resource of that node as to the number of simulation objects currently being served. This number plus one is the appropriate index number. The index is passed to an instance of Database, which retrieves the corresponding frequency data from a file in secondary storage. (The frequency data is stored there during runs of NetworkSimulation.) The data is then passed to an instance of DiscreteApproximation to generate the random deviate.

REFERENCES

Buchholz, P. (1989). Definition of submodels and classification of aggregates (Integrated Modelling Support Environment project report, R5.4-1 Version 2). University of Dortmund.

Chandy, K., Herzog, U., & Woo, L. (1975). Parametric analysis of queuing networks. IBM Journal of Research & Development, 19, 43-49.

Coad, P., & Yourdon, E. (1990). Object-oriented analysis. Englewood Cliffs, NJ: Prentice-Hall.

Fishwick, P. (1986). Hierarchical reasoning: simulating complex processes over multiple levels of abstraction (UF-CIS Technical Report TR-86-6). Gainesville, FL: University of Florida, Computer and Information Sciences Department.

Franta, W. (1977). The Process View of Simulation. Amsterdam: North-Holland.

Garrison, W. (1990). Network II.5 tutorial - network modeling without programming. In O. Balci, R. Sadowski, & R. Nance. (Eds.), Proceedings of the 1990 Winter Simulation Conference (pp. 132-135). New Orleans, LA: Society for Computer Simulation.

Goldberg, A., & Robson, D. (1989). SMALLTALK-80: the language. Reading, MA: Addison Wesley.

Marie, R. (1979). An approximate analytical technique for general queuing networks (IEEE Transactions on Software Engineering, SE-5 (5)), pp. 530-538.

O'Reilly, J., & Whitford, J. (1990). SLAM II tutorial. In O. Balci, R. Sadowski, & R. Nance, (Eds.), Proceedings of the 1990 Winter Simulation Conference (pp. 72-76). New Orleans, LA: Society for Computer Simulation.

Rothenberg, J., Narain, S., Steeb, R., Hefley, C., & Shapiro, N. (1989). Knowledge-based simulations: an interim report (RAND Note N-2897-DARPA). Santa Monica, CA: Rand Corporation.

Taha, H., Taylor, R., & Younis, N. (1990). Simulation and animation with SIMNET II and ISES. In O. Balci, R. Sadowski, & R. Nance, (Eds.), Proceedings of the 1990 Winter Simulation Conference (pp. 99-105). New Orleans, LA: Society for Computer Simulation.

Zeigler, B. (1984). Multifacetted modelling and discrete-event simulation. New York: Academic Press.

Appendix A

Variable Definitions by Object Class

Format:

> **Class**
> Class Variables
> instance variables

(Class variables will begin with a capital letter; instance variables will not)

Aircraft

currentNode - the ID number of the current **TaskNode** of the aircraft.

neededParts - a **Bag** containing the names of the aircraft parts needing repair on a particular aircraft.

Aircraft(i)

AircraftType - identifies the name of the class to which an aircraft belongs.

ClassTurnNetwork - The postflight and preflight aircraft servicing network.

RepairableParts - a **Dictionary** containing the names of parts that may need repair and their probability of failure.

Arcs

head - the ID of the **TaskNode** at the "head" of the arc.

tail - the ID of the **TaskNode** at the "tail" of the arc.

probability - the probability of the arc being chosen as the traversal path upon leaving the "tail" **TaskNode.**

Database

file

recordCount

currentIndex

fieldList

recordSize

headerSize

currentRecord

DiscreteApproximation

start - the lower bound of the probability domain.

stop - the upper bound of the probability domain.

step - the cell width of the approximation.

data - a "Stream" of frequency values for each cell.

Entity

ClassTaskNetwork - the **TaskNetwork** to be traversed by the entities.

entryTime - the simulation time at which the entity enters the simulation.

entityID - the ID number of the entity.

currentNode - the node at which the entity currently resides.

IndexedApproximation

prob - random deviate returned to simulation.

resourceType - a member of dummyResourceSet.

NetworkHistogram

resolution - the ratio of cell entries to unit markers in the histogram plot.

sumOfSquares - used to calculate sample standard deviation of network traversal times.

nument - the number of entities cycling through the network.

serviceMean - the theoretical mean service time (excluding queueing time) for the entire network.

serviceDev - the theoretical standard deviation of the above.

Network Simulation

statistics - contains the network histogram.

nument - the number of entities cycling through the network.

Parts

currentNode - the ID number of the current task node of the part.

Part(i)

ClassTaskNetwork - the part repair network.

PartType - identifies the name of the class to which a part belongs.

Simulation

dummyResourceSet - a **Set** of the names of the simulation resources that represent aggregated networks.

SimulationObject

entityID - a unique assignable ID code.

TaskNetwork

partLocations - a **Dictionary** that provides the current **TaskNode** ID number for each **Part** ID number.

aRandom - contains the current random number being used for probabilistic branching in traversing the network.

root - the first **TaskNode** in a network.

taskID - an ID number to identify a **TaskNetwork** when it is used as a **TaskNode** on another network.

taskType - the name of the simulation **Resource** used to accomplish the task; in this case, where the network is a node on another network, the resource will be 'Network.'

arcsOut - a **Set** of instances of **Arcs** that defines the possible temporal paths.

TaskNode

taskID - an ID number for a node.

taskType - the name of the simulation **Resource** used to accomplish the task.

duration - the time required to complete a task (may be a value or a probability distribution).

arcsOut - a **Set** of instances of **Arcs** that defines the possible temporal paths.

Appendix B

Key Methods by Class

AircraftSimulation

defineDummyResources - used to specify any aggregated or "dummy" resources in the simulation model. Uses the statement:

```
self produce:   aDummyResource
```

There may be three types of these statments: the resource at a dummy node, a resource representing a network, and a resource representing an aggregated network.

initialize - this method already exists in **Simulation**. Here it is modified by inclusion of the statement:

```
dummyResourceSet:   Set new
```

Aggregate

newTaskNode - This method specializes the method in the **TaskNode** superclass to give the node a "duration" as specified by **IndexedApproximation**.

```
newNode ←  super newTaskNode.
newNode duration: (IndexedApproximation type: 'Aggregate').
newNode type: 'Aggregate'.
↑ newNode
```

Aggregate(i)

new - This method is sent to the class to create an instance of the appropriate aggregated network. It is a modified version of the "new" method for **TaskNetwork**. The aggregated network will consist of a single **TaskNode** from the subclass, **Aggregate**.

```
aTaskNetwork  ←  super new.
aNode    Aggregate newTaskNode.
aNode taskID: '1'.
aTaskNetwork firstTask: aNode.
↑  aTaskNetwork
```

Entity

startUp: anEntityID - The "startUp" method in **SimulationObject** has been modified to help implement the "recycling" of entities in the **TaskNetwork**. The modified method allows a new **Entity** to appear to the simulation as being the same as the **Entity** just leaving the simulation. This happens by recycling the ID number:

```
entityID ← anEntityID
```

IndexedApproximation

next - as with all members of the system class, **ProbabilityDistribution,** this method returns a random deviate of the **DiscreteApproximation** corresponding to the appropriate index. The index is obtained by querying the simulation as to the number of other simulation objects using the resource, "resourceType" (an instance variable of the IndexedApproximation) as follows:

```
index ← Simulation active resourceSet occurrencesOf: (self resourceType)
```

The message, "next", is then sent to the **DiscreteApproximation** that corresponds to index.

NetworkSimulation

defineArrivalSchedule - used to create "nument" number of entities at simulation time 0:

```
(1 to: nument by: 1)
do: [ count | self schedule: [Entity new startUp: count] at: 0 ]
```

initialize - this method from **Simulation** has been modified to initialize the histogram parameters, set nument, and set the **TaskNetwork** to be traversed.

```
statistics ← NetworkHistogram from: astart to: astop by: astep withResolution:
aResolution
nument ← aNumber
Entity initializeWithNetwork: aTaskNetwork
```

exit: aSimulationObject - this method from **Simulation** has been modified so a new **SimulationObject** is created that is a "clone" of the **SimulationObject** exiting the simulation. The cloned **SimulationObject** enters the simulation at the exact time the original **SimulationObject** leaves. The modified method also disables the collection of traversal time data on the first traversal cycle of the **SimulationObject** to prevent biasing the throughput statistics. The following statements were added:

```
(aSimulationObject entryTime) > 0.0
    ifTrue: [statistics store: currentTime - aSimulationObject entryTime]
self schedule: [Entity new startUp: (aSimulationObject entityID)] at: (self time)
```

Simulation

dummyResourceSet/dummyResourceSet: - used to retrieve or set the contents of the **Simulation** instance variable, dummyResourceSet.

SimulationObject

acquire: amountofResource: resourceName - this existing method has been modified to enable the apparent behavior of aggregated resources. When a **SimulationObject** attempts to acquire an aggregated resource, the simulation actually produces another occurrence of that resource:

```
    (Simulation active dummyResourceSet includes: resourceName)
     ifTrue: [Simulation active produce: amountOf: resourceName]
```

TaskNetwork

duration: aSimulationObject - simulates the traversal of the network by aSimulationObject and returns the time required to traverse the network.

mean - returns the expected total of the service times during a traversal of the network

next: aPartID - returns the next node in the network to be traversed by the part identified by *aPartID*. This may be the result of a probabilistic branching.

variance - returns the variance of the total service time during a traversal of the network

TaskNode

duration: aSimulationObject - returns a number representing the time required to use the resource at the node. It may be a fixed number or a random deviate.

mean - the mean of the time required to use the resource at the node. In the case of a fixed time it returns the fixed number.

variance - the variance of the time required to use the resource at the node. In the case of a fixed time it returns 0.